Parcheado del kernel conado del kernel con PREEMPT_RT

En esta práctica vamos a aplicar el parche PREEMPT_RT al kernel de nuestra Raspberry Pi y configurar el kernel en modo Fully Preemptible Kernel (Real-Time).

1 Requisito previo

Vamos a utilizar el mismo entorno de trabajo que hemos utilizado en el curso de Desarrollo de Drivers en Linux.

Además debemos preparar el sistema como indica la actividad de configuración del sistema.

2 Parcheado del kernel con preemt rt

Una vez configurado el sistema podemos pasar a parchear el kernel. Seguiremos los pasos indicados en esta sección. Trabajaremos en un directorio reservado para la compilación del kernel, por ejemplo:

```
$ mkdir ~/rpi-kernel
$ cd ~/rpi-kernel
```

Por supuesto, se puede elegir cualquier otra ruta, pero en los apartados siguientes nos referiremos a esta ruta como el directorio de trabajo, y a la máquina que usamos para la compilación cruzada como el host.

A continuación segimos estos pasos:

1. Decargamos las fuentes del kernel que viene con la distribución Raspberry Pi OS de 32 bits (la recomendada por defecto).

   Para esto creamos un directorio linux y creamos en él un repo git en el que descargamos exactamente la versión del kernel usada.

   Para saber la versión del kernel que necesitamos, ejecutamos el siguiente comando en la raspberry pi:

```
$ uname -r
5.10.92-v7+
```

Consultamos la web del kernel https://github.com/raspberrypi/linux, seleccionamos el branch rpi-5.10.y y el fichero Makefile. Pinchamos en History y buscamos la versión que coincida con la versión del kernel instalada.

En el momento de preparar este documento, con el sistema actualizado (sudo apt full-upgrade), la versión del kernel es 5.10.92-v7+. Por tanto bucamos en History la versión 5.10.92. Vemos que la entrada del 16 de enero está marcada con 5.10.92. Ésta es una versión upstream del kernel, sin las modificaciones para la raspberry pi, copiamos el sha de la del 17 de Enero (650082a559a570d6c9d2739ecc62843d6f951059), que tiene el merge con el remote tracking branch 'stable/linux-5.10.y'.

Con esta información ya podemos proceder a descargar el código:

```
$ mkdir linux
$ cd linux
$ git init
$ git remote add origin https://github.com/raspberry/linux
$ git fetch --depth 1 origin 650082a559a570d6c9d2739ecc62843d6f951059
$ git checkout -b 5.10.92 FETCH_HEAD
```

2. Creamos un branch para el kernel parcheado

```
$ git checkout -b 5.10.92-rt
```

3. Descargamos el parche de preempt rt. Buscamos la versión para el kernel que hemos descargado en https://cdn.kernel.org/pub/linux/kernel/projects/rt/5.10/. Como vemos, no hay una versión para la 5.10.92, pero sí hay una para la versión 5.10.90, descargamos esa.

```
$ cd ..
$ wget https://cdn.kernel.org/pub/linux/kernel/projects/rt/5.10/patch-5.10.90-rt61-rc1.
```

4. Parcheamos el kernel

```
$ cd linux
$ xzcat ../patch-5.10.90-rt61-rc1.patch.xz | patch -p1
```

Con este parche no falla ningún hunk, estamos de suerte.

5. Hacemos el commit de los cambios

```
$ git add -A
$ git commit -m "patch-5.10.90-rt61-rc1.patch.xz appliead"
```

6. Instalamos los compiladores cruzados (si no están ya instalados):

```
$ sudo apt install crossbuild-essential-armhf
```

7. Creamos el fichero export.sh en el directorio de trabajo (~/rpi-kernel/ en nuestro ejemplo) con variables de entorno para facilitar la compilación del kernel:

```
#!/bin/bash

SCRIPT_DIR=$(cd -- "$(dirname -- ${BASH_SOURCE[0]})" &> /dev/null && pwd)

export ARCH=arm
export CROSS_COMPILE=arm-linux-gnueabihf-
export KERNEL=kernel7
```

8. Aplicamos la configuración por defecto y luego configuramos el kernel seleccionando: General Setup -> Preemption Model -> Fully Preemptible kernel (Real-Time).

```
$ source ../export.sh
$ make mrproper
$ make bcm2709_defconfig
$ make menuconfig
```

9. Compilamos el kernel (esto tarda un rato):

```
$ make -j4 zImage modules dtbs
```

10. Una vez compilado vamos a crear un tarball con los ficheros a instalar. Para ello creamos un directorio kernel-rt en el directorio de trabajo y lo seleccionamos como target para la instalación. Los dtbs irán en un subdirectorio boot.

```
$ mkdir ../kernel-rt
$ make INSTALL_MOD_PATH=../kernel-rt/ modules_install
$ make INSTALL_DTBS_PATH=../kernel-rt/boot dtbs_install
$ cp arch/arm/boot/zImage ../kernel-rt/boot/kernel7-rt.img
```

11. Creamos el tarball cambiando el uid y el gid a 0 (root):

```
$ cd ../kernel-rt
$ rm lib/modules/5.10.92-rt61-rc1-v7+/build
$ rm lib/modules/5.10.92-rt61-rc1-v7+/source
$ fakeroot -- tar cvzf ../kernel-rt.tar.gz *
```

12. Copiamos el tarball a la raspi (sustituir la ip de la raspberry):

```
$ scp ../kernel-rt.tar.gz pi@<ip raspi>:
```

13. Nos conectamos a la raspi y descomprimimos el fichero sobre el direcotrio raiz (sustituir la ip de la raspberry):

```
$ ssh pi@<ip raspi>
pi@raspberry$ sudo tar xvzf kernel-rt.tar.gz --keep-directory-symlink -C /
```

14. Editamos el fichero /boot/config.txt y añadimos la siguiente línea al final:

```
kernel=kernel7-rt.img
```

15. Actualmente el driver dwc_otg tiene un bug que hace que se cuelque el sistema cuando se usan *threaded interrupts* (rtis ejecutadas en hilos), cuando uno de estos hilos es interrumpido con un spinlock fiq cogido (https://www.osadl.org/Single-View.111+M5c03315dc57.0.html). Una solución de compromiso es configurar el driver para que no use fiq, añadiendo a la línea de comandos del kernel (cmdline.txt) las siguientes opciones: *dwc_otg.fiq_fsm_enable=0 dwc_otg.fiq_enable=0 dwc_otg.nak_holdoff=0*.

16. Reiniciamos la raspberry pi y comprobamos que tenemos el kernel de tiempo real:

```
pi@raspberry$ sudo shutdown -r now
... tras el reinicio ...
pi@raspberry$ uname -rv
5.10.92-rt61-rc1-v7+ #1 SMP PREEMPT_RT Tue Mar 1 10:56:23 CET 2022
```